Attorney's Docket No.: 10559-607001 / P12891

APPLICATION

FOR

UNITED STATES LETTERS PATENT

10054759.011702

TITLE: MODELING A LOGIC DESIGN

APPLICANT: WILLIAM R. WHEELER AND TIMOTHY J. FENNELL

CERTIFICATE OF MAILING BY EXPRESS MAIL

Express Mail Label No EU047038729US

I hereby certify that this correspondence is being deposited with the United States Postal Service as Express Mail Post Office to Addressee with sufficient postage on the date indicated below and is addressed to the Commissioner for Patents, Washington, D C 20231

1-17-02

Date of Deposit

Signature

Gabriel Lewis

Typed or Printed Name of Person Signing Certificate

MODELING A LOGIC DESIGN

TECHNICAL FIELD

This invention relates to modeling a logic design using functional block diagrams and to generating simulation code that corresponds to the logic design.

BACKGROUND

Logic designers typically model logic designs, which may include circuit elements such as flip-flops, registers, and logic gates, using block diagrams. Computer-aided design (CAD) systems may be used to generate such block diagrams electronically. Conventional CAD systems, however, do not provide the flexibility and types/extent of information desired by many logic designers.

Moreover, models created using conventional CAD systems are often of little assistance when simulating the logic design. Heretofore, a logic designer had to make a separate "simulation" model of the logic design using a simulation code, such as Verilog and Very High-Level Design Language (VHDL). The simulation model can be cumbersome and difficult to understand, particularly for complex logic designs.

DESCRIPTION OF THE DRAWINGS

Fig. 1 is a flowchart showing a process for modeling a logic design using functional block diagrams and generating simulation code that corresponds to the logic design.

Fig. 2 is a block diagram of a menu for selecting functional block diagrams for the logic design.

Fig. 3 shows functional block diagrams that were selected from the menu.

Fig. 4 shows the functional block diagrams of Fig. 3 interconnected using virtual wire.

Fig. 5 is a block diagram of a computer system on which the process of Fig. 1 may be executed.

DESCRIPTION

Referring to Fig. 1, a process 10 is shown for modeling a logic design. Process 10 may be implemented using a computer program running on a computer or other type of programmable machine, as described in more detail below.

In operation, process 10 displays (101) a menu, such as menu 12 shown in Fig. 2. Menu 12 includes options for use in creating a graphical representation of a logic design. These options correspond to functional block diagrams for various circuit elements, such as registers 14, ports 16, AND gates

10045479 .011502

18, OR gates 20, buffers 22, multiplexers 24 (MUX), and so forth. Data, including computer code, that defines the functional block diagrams for these circuit elements is stored in a database. The data defines inputs and outputs of each functional block diagram, as well as the operation to be performed on the inputs by the functional block diagram to generate the outputs. In one embodiment, the functional block diagrams are software "objects". By way of example, in the case of an "AND" gate, the data specifies that the functional block diagram includes N (N>1) inputs, one output, and the definition of an operation to be performed on the inputs to generate the output. In the case of state elements, such as registers and flip-flops, the inputs may include one or more clock signals.

The options on menu 12 also include a combinational (COMBO) box option 26. COMBO box option 26 provides an undefined functional block diagram for use in a logic design. The undefined functional block diagram may be defined by the user to simulate any circuit element or combination of circuit elements. The user may enter simulation code via a graphical user interface (GUI) (not shown) to define the functionality of an undefined functional block diagram. The simulation code may specify inputs, outputs and operations to

be performed on the inputs to generate the outputs. Examples of simulation code that may be used include, but are not limited to, Verilog, C++ and VHDL.

Process 10 receives (102) an input selection from menu 12. That is, a designer selects one or more of the options from menu 12. The selection is transmitted to process 10, which retrieves (103), from the database, a functional block diagram that corresponds to the selection. For example, a designer may select register option 14. In response, process 10 retrieves a "register" functional block diagram from the database. If the designer selects COMBO box option 26, process 10 retrieves an undefined functional block diagram from the database. The designer specifies the function of that block diagram using, e.g., simulation code.

Process 10 creates (104) a graphical representation of a logic design using retrieved (103) functional block diagrams. That is, process 10 displays the retrieved functional block diagrams and the designer arranges the functional block diagrams to represent a logic design. Although the designer is moving the block diagrams by, e.g., dragging and dropping, process 10 arranges (104a) the block diagrams in the sense that their movement is executed and stored via process 10.

Fig. 3 shows functional block diagrams 30 that have been arranged prior to being interconnected.

Once the functional block diagrams are arranged, process 10 interconnects (104b) the block diagrams using virtual wires. That is, the designer selects wire option 22 from menu 12 and connects the inputs and/or outputs thereof using the virtual wires. Process 10 stores the configuration of the logic design, including the virtual wire connections, in memory. Fig. 4 shows the functional block diagrams of Fig. 3 following interconnection. It is noted that process 10 may display the definitions (e.g., 34, 36 and 38) of each input or output terminal, or not, as desired.

If there are any problems with the interconnections (107), process 10 displays a visual indication of the problem(s) with the design. In this regard, process 10 automatically runs a diagnostic on the logic design to confirm that the logic design comports with a set of predefined rules specifying, e.g., proper connections between terminals on different functional block diagrams. Examples of connection problems include, but are not limited to, unterminated connections and outputs running into the wrong inputs (e.g., a logic gate output running into a clock terminal input).

In this embodiment, process 10 illuminates the logic design in red if there is a problem. Other indicators may be provided instead of, or in addition, to, illuminating the logic design in red. For example, the indication may specify the nature of the problem in words or graphics and its location within the logic design.

If there are any problems with the displayed logic design, process 10 returns to one of the previous blocks 101. 102, 103, and 104, where the problem may be corrected.

Assuming that there are no problems with the design, or that the problems have been corrected, process 10 generates (105) simulation code for the design. In this embodiment, process 10 generates Verilog, VHDL, and/or C++ simulation code. However, the simulation code is not limited to generating only these two types of simulation code.

Generally speaking, the designer may select, e.g., via a GUI (not shown), which simulation code (C++, VHDL, Verilog) process 10 will generate. The type of simulation desired may dictate the simulation code that process 10 will generate.

Process 10 generates the simulation code knowing the functional block diagrams that make up the logic design, their inputs and outputs, and their interconnections. For each functional block diagram, process 10 generates

appropriate simulation code and provides the appropriate inputs and outputs. Process 10 combines the generated simulation code for the various functional block diagrams into simulation code that defines the logic design.

Once simulation code for the logic design has been generated (105), process 10 tests (106) the logic design. This may be done by propagating one or more states through the simulation code and determining if there is an error based on the state propagation. For example, process 10 may propagate a logical one (1), a logical zero (0), and/or an undefined (X) state through the simulation code. If the resulting output of the simulation code is not what is expected, process 10 will indicate to the logic designer that an error exists in the logic design. The designer may then go back and change the logic design, as desired.

Fig. 5 shows a computer 40 on which process 10 may be executed. Computer 40 includes a processor 42, a memory 44, and a storage medium 46 (e.g., a hard disk) (see view 48). Storage medium 46 stores data 50 that defines a logic design, a database 52 that includes the functional block diagrams, simulation code 54 (e.g., C++, Verilog, VHDL) for each functional block diagram and for the resulting logic design,

and machine-executable instructions 56, which are executed by processor 42 out of memory 44 to perform process 10.

Process 10, however, is not limited to use with the hardware and software of Fig. 5; it may find applicability in any computing or processing environment. Process 10 may be implemented in hardware, software, or a combination of the two. Process 10 may be implemented in one or more computer programs executing on programmable computers or other machines that each includes a processor, a storage medium readable by the processor (including volatile and non-volatile memory and/or storage elements), at least one input device, and one or more output devices. Program code may be applied to data entered using an input device, such as a mouse or a keyboard, to perform process 10.

Each such program may be implemented in a high level procedural or object-oriented programming language to communicate with a computer system. However, the programs can be implemented in assembly or machine language. The language may be a compiled or an interpreted language.

Each computer program may be stored on an article of manufacture, such as a storage medium or device (e.g., CD-ROM (compact disc read-only memory), hard disk, or magnetic diskette), that is readable by a general or special purpose

programmable machine for configuring and operating the machine when the storage medium or device is read by the machine to perform process 10. Process 10 may also be implemented as a machine-readable storage medium, configured with a computer program, where, upon execution, instructions in the program cause the machine to operate in accordance with process 10.

The invention is not limited to the specific embodiments set forth above. For example, process 10 is not limited to the types and content of displays described herein. Other displays and display contents may be used. Process 10 is not limited use with the simulation languages noted above, e.g., Verilog, VHDL, and C++. Process 10 also is not limited to the order of execution set forth in Fig. 1. That is, the blocks of process 10 may be executed in a different order than that shown to produce a desired result.

Other embodiments not described herein are also within the scope of the following claims.

What is claimed is: